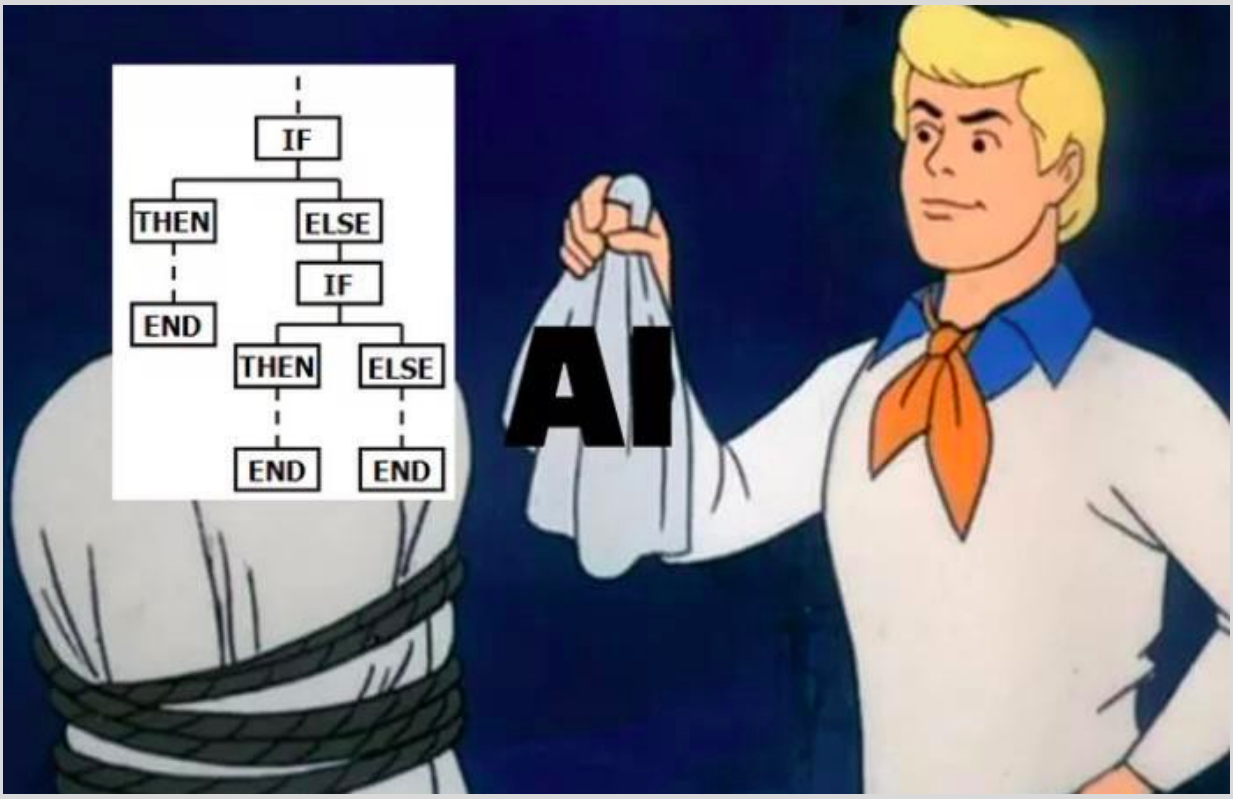
Veštačka inteligencija 1

Skripta za usmeni ispit



27.01.2023. Uroš Đerić 46/19 IT

Ispitna pitanja

1	Agenti i okruženje. Racionalni agent. Osobine okruženja.	Poglavlje 2
2	Agenstki program i tipovi agentskih programa.	
3	Definicija problema u VI. Pronalaženje rešenja i stablo pretraživanja. Strukture podataka u algoritmima pretraživanja. Merenje performnsi algoritma pretraživanja.	Poglavlje 3
4	Strategije za slepo pretraživanje.	
5	Strategije za informisano (heurističko) pretraživanje	
6	Heurističke funkcije.	
7	Algoritmi za lokalno pretraživanje	Poglavlje 4
8	Pretraživanje sa nedeterminističkim akcijama	
9	Pretraživanje u okruženju koje se ne može potpuno sagledati	
10	Pretraživanje u igrama. MINIMAX stablo pretraživanja. Optimalno odlučivanje u igrama sa dva i više igrača.	Poglavlje 5
11	Alfa-beta odsecanje kod MINIMAX algoritma.MINIMAX do određene dubine.	
12	Logička posledica i pravila izvođenja. Predstvaljanje znanja u iskaznoj logici. Logičke ekvivalencije, valjane i zadovoljive formule.	Poglavlje 6
13	Sintaksa i semantika logike prvog reda. Simboli i interpretacije. Kvantifikatori. Baza znanja sa logikom prvog reda.	
14	Koraci razvoja projekta u oblasti inženjerstva znanja.	
15	Zaključivanje u logici prog reda.	
16	Ontološki inženjering. Kategorije i objekti. Sistemi rasuđivanja za kategorije.	
17	Izražavanje nesigurnosti. Osnove teorije verovatnoće	Poglavlje 7
18	Bajesovo pravilo i Bajesova mreža.	
19	Verovatnoća u nedeterminističkom okruženju. Očekivana korisnost. Funkcija korisnosti sa više atributa.	Poglavlje 8
20	Mreže odlučivanja. Ekspertni sistemi zasnovani na teoriji odlučivanja.	
21	Pojam mašinskog učenja. Vrste mašinskog učenja. Nadgledano učenje i hipoteze	Poglavlje 9
22	Klasifikacija: metoda zasnovana na instancama, učenje stabla odlučivanja.	
23	N-gram modeli prirodnog jezika. Klasifikacija teksta.	Poglavlje 10
24	Pribavljanje informacija. Ekstrakcija informacija.	
25	Formalni jezici. Stohastička kontekstno-slobodna gramatika.	
26	Sintaksna analiza. Gramatika i logika.	

1. Agenti i okruženje. Racionalni agent. Osobine okruženja

Agent je neko ili nešto što se ponaša na određeni način u skladu sa informacijama koje dobija i obradi. Te informacije dobija koristeći senzore ili čula (ako je ljudski agent). Na osnovu prikupljenih i obrađenih informacija agent izvršava određene akcije. Primera radi, ljudski agent će, koristeći oči, videti da predmet leti ka njemu i izvršiće akciju izbegavanja tog predmeta. Softverski agent će recimo po primljenoj poruci sa tastature da je obradi tj. očita i da pošalje adekvatan odgovor korisniku. Robotski agent će, koristeći senzor, da očita da nailazi na zid i izvršiće akciju pomeranja.

Agentska funkcija mapira bilo koji niz zapažanja na akciju. Ona se može opisati i preko tabele, ali bi za većinu realnih agenata ta tabela bila beskonačna. Takva tabela predstavlja eksternu karakterizaciju agenta. Interno, agenstka funkcija za veštačkog agenta implementira se kao agentski program. **Dakle, agentska funkcija je apstraktni matematički opis, a agentski program je konkretna implementacija, izvršavanje na nekom fizičkom sistemu**.

Racionalni agent je agent koji čini ispravnu stvar (ispravno je popunjena tabela agentske funkcije) sa maksimalnom merom učinka. Drugim rečima, **za svaki niz zapažanja, racionalni agent treba da se odluči za akciju za koju se očekuje da maksimizira njegov učinak, uzimajući u obzir niz zapažanja i njegovo ugrađeno znanje**. Mera učinka zapravo predstavlja stepen poželjnosti. Ona se različito definiše za različite sisteme. Mera racionalnosti agenta se gleda na osnovu stanja okruženja, a ne agenta.

Problemi koje agenti treba da reše u nekom okruženju definišu se preko četiri parametra: **učinak, okruženje, aktuatori i senzori**. Okruženje, u zavisnosti od senzora i uslova očitavanja, može biti **potpuno**, **parcijalno sagledljivo** i **nesagledljivo**.

U jednom okruženju može biti više agenata, ali nekad se ne može tačno odrediti broj agenata. **Okruženja sa više agenata mogu biti konkurentna i kooperativna**.

Okruženje može biti determinističko ili stohastičko. Ako je sledeće stanje okruženja potpuno određeno tekućim stanjem i akcijom koju agent izvrši tada kažemo da je okruženje determinističko, u suprotnom kažemo da je stohastičko.

Termin **stohastičko** se koristi kada nepoznavanje mogućeg ishoda izražavamo verovatnoćom, dok termin **nedeterminističko** koristimo za okruženja gde se navode samo rezultati akcija bez navođenja verovatnoća i tada agent treba da bude uspešan u svim mogućim slučajevima.

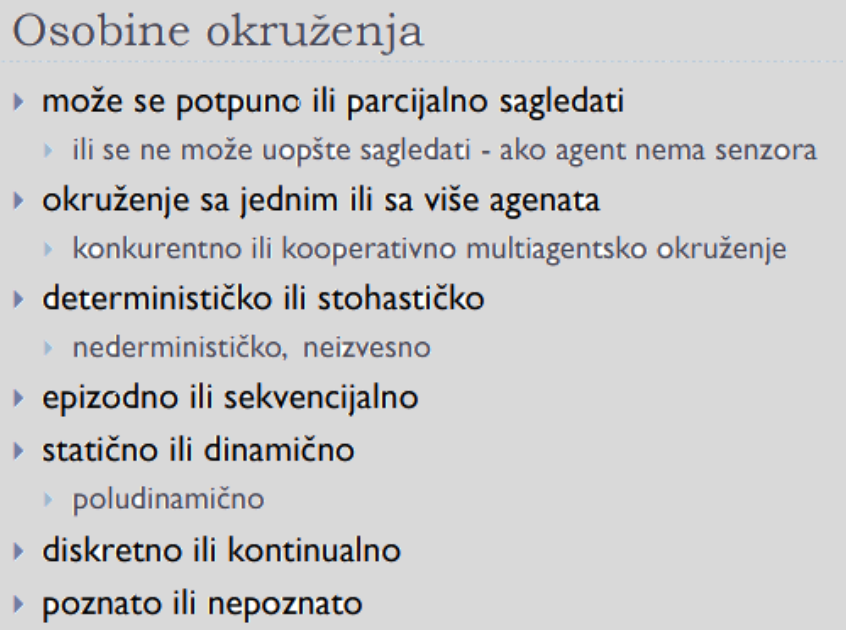
Za okruženje koje nije deterministično, a ne možemo potpuno da ga sagledamo kažemo da je **neizvesno**.

Okruženje može da bude **epizodično** i **sekvencijalno**. Epizodična su kada agent prima neka zapažanja i vrši određenu akciju iz više navrata ili epizoda. Akcije u jednoj epizodi ne zavise od akcija iz prethodnih. Primer: Robot koji razvrstava delove gde za svaki deo može da odluči da li je potreban ili ne. S druge strane imamo sekvencijalna okruženja i tu akcije imaju uticaj na buduće akcije. Primer toga bi bio šah gde pomeranjem pijuna potencijalno oslobađamo ili zagradimo neku drugu figuru.

Okruženje može biti **statično** i **dinamično**. Ako se okruženje može promeniti dok agent odlučuje o akciji onda je ono dinamično - u suprotnom je statično. Lakše je delati u statičnom okruženju jer agent onda ne mora konstantno da odlučuje o akciji. Ukoliko se u nekom periodu vremena okruženje nije promenilo, a jeste se promenio agentov učinak govorimo o **poludinamičnom** okruženju.

Razlika između **diskretnog** i **kontinualnog** okruženja izražava se kroz stanje okruženja u toku vremena i zapažanja i akcije agenta. Na primer, šah ima konačan broj različitih stanja (ako nemamo sat) i ima diskretni skup zapažanja i akcija. Vožnja taksija ima kontinualne promene stanja.

Finalno, okruženje može biti **poznato** i **nepoznato**. U nepoznatom okruženju agent mora da uči kako stvari funkcionišu da bi mogao da donosi prave odluke dok u poznatom to ne mora da radi.



2. Agentski program i tipovi agentskih programa

Agentski program predstavlja program čiji je ulaz *trenutno* zapažanje a izlaz odgovarajuća akcija koja teži da bude ispravna.

Postoje četiri tipa agentskih programa koji definišu sve osnovne principe inteligentnih sistema:

- jednostavni refleksni agenti
- refleksni agenti zasnovani na modelu
- cilj-orijentisani agenti
- korisno-orijentisani agenti

Svaki od navedenih se mogu transformisati u agente koji uče (learning agents)

- Jednostavni refleksni agenti

Ovo je najjednostavnija vrsta agenata koja selektuje akciju na osnovu trenutnog zapažanja, ignorišući istoriju zapažanja. Jednostavan primer ovakvog agenta je usisivač.

- Refleksni agenti zasnovani na modelu

Znanje koje opisuje kako svet funkcioniše naziva se model, a agenti koji koriste to znanje nazivaju se agenti zasnovani na modelu. Često se koriste kada imamo okruženje koje je samo parcijalno sagledljivo. Ovi agenti imaju dve vrste znanja. Prvo, mora znati kako se okruženje menja nezavisno od agenta, a drugo jeste znanje kako akcije agenta menjaju okruženje.

- Cilj-orijentisani agenti

Agent koji usvaja dodatno znanje, informaciju o cilju. Recimo za taksistu, ispravan potez kada priđe raskrsnici zavisi od toga gde mu je cilj. Iako manje efikasni, mnogo su fleksibilniji budući da njihovo znanje može biti lako promenljivo bez peripetija sa uslov-akcijama.

- Korisno-orijentisani agenti

Predstavlja nadogradnju u odnosu na ciljno-orijentisane agente. Uz cilj, agent takođe traži najpovoljniju rutu do svoje destinacije po određenim kriterijumima (najjeftiniji, najbrži,...).

Koristeći **funkciju učinka** (interna reprezentacija mere učinka), agent odlučuje koja mu je najpogodnija ruta u zavisnosti od cilja. Ovaj agent se koristi pre ciljno-orijentisanog najčešće u dva slučajeva: kada imamo poprečne ciljeve i samo jedan može biti postignut (na primer brzina i sigurnost) gde funkcija učinka pravi neki kompromis. Drugi slučaj je kada imamo više ciljeva od kojih se ni jedan ne može sa sigurnošću dostići, tada funkcija učinka obezbeđuje da se verovatnoća uspeha iskaže u odnosu na značaj cilja.

- Agenti koji uče

Agenti koji uče sadrže četiri konceptualne komponente, najbitnija je razlika između **elementa učenja** (learning elements) i **elementa izvršavanja** (performance element). Zadatak elementa učenja je da pravi poboljšanja, dok se element izvršavanja bavi **selektovanjem akcija**. Element učenja koristi povratne informacije od **kritike** (Critic) koja ocenjuje agentovo ponašanje i određuje kako se element izvršavanja može unaprediti da radi bolje. **Kritika** je neophodna jer zapažanja ne obezbeđuju informacije o uspehu agenta. Tu je i **generator problema** čija je uloga da predlaže akcije koje će dovesti do novih, informativnih iskustava. Na ovaj način agent može da izvršava neke akcije koje nisu kratkoročno efikasne ali potencijalno mogu da dovedu dugoročno do optimalnijih stanja.

3. Definicija problema u VI. Pronalaženje rešenja i stablo pretraživanja. Strukture podataka u algoritmima pretraživanja. Merenje perfomansi algoritma pretraživanja.

Kako bi se rešio problem bilo koje kompleknosti u VI, potrebno ga je jasno i formalno definisati. Problem se formalno može definisati pomoću 5 komponenti:

- Inicijalno stanje - stanje u kom se agent nalazi na početku.
- Opis akcije koje su dostupne agentu u određenom stanju (ACTIONS(s))
- Tranzicioni model (Result(s,a))- opis rezultata akcije, odnosno šta akcija radi, kako akcija menja stanje.
- Funkcija cilja
- Cena putanje

Rešenje problema se definiše kao niz akcija koje vode do cija, a postoji i mera kvaliteta tog rešenja koja se izražava cenom putanje. Optimalno rešenje je ono rešenje koje ima najmanju cenu putanje.

Rešenje se traži u vidu sekvenci akcija, tako da se algoritam pretraživanja svodi na razmatranje raznih mogućih sekvenci akcija. Moguće sekvence akcija koje počinju u inicijalnom stanju formiraju takozvano **stablo pretraživanja** čiji je koren to inicijalno stanje. **Grane** u stablu pretraživanja su akcije, a **čvorovi** su stanja iz prostora stanja. **Otvaranje (expand) čvora** u stablu pretraživanja predstavlja primenu svih akcija koji su dostupni iz stanja koje je predstavljeno čvorom. Otvaranje čvora podrazumeva generisanje novih čvorova, odnosno novih stanja. **Način izbora čvorova za otvaranje definiše strategiju pretraživanja.** Da bi se izbegle zatvorene putanje potrebno je zapamtiti u kojim stanjima smo bili koristeći zatvorene liste. **Zatvorena lista je struktura podataka koja se koristi da upamti svaki otvoreni čvor**. Kada se izgeneriše novi čvor, proverava da li se takav već nalazi na listi i doda ga na listu ukoliko je nov ili eliminiše ukoliko već postoji. Taj način pretrage se naziva **Graph search.** Razlikuje se od tzv. *Tree search*-a po tome što **pamti stanja u kojima je bio**.

Algoritmima za pretraživanje je potrebna struktura podataka za opis stabla pretraživanja. Za svaki čvor n imamo strukturu podataka koja sadrži četiri komponente:

- n.State - stanje iz prostora stanja
- n.Parent - čvor u stablu pretraživanja koji je generisao čvor
- n n.Action - akcija kojom je generisan čvor n
- n.Path-Cost - cena putanje, obeležava se i sa g(n).

Čvor je struktura podataka koja sadrži informacije o roditelju, deci, dubini, vrednosti putanje, **a stanje se odnosi na osobine okruženja**. **Jedan čvor se može odnositi na više stanja**, ako je stanje generisano na različitim putanjama u stablu pretraživanja.

Pri merenju perfomansi algoritama za pretraživanje posmatraju se četiri osobine:

- kompletnost - da li de algoritam sigurno pronaći rešenje ako ono postoji?
- optimalnost - da li de pronaći optimalno rešenje?
- vremenska složenost - koliko vremena je potrebno da se pronađe rešenje?
- prostorna složenost - koliko memorije je potrebno za pretraživanje?

Zbog načina predstavljanja stabla pretraživanja, u VI se za izražavanje složenosti koriste:

- b - faktor grananja (maksimalan broj naslednika)
- d - dubina najplićeg cilja (broj koraka do korena do cilja koji jemu je najbliži)
- m - maksimalna dužina bilo koje putanje grafa stanja.

Vremenska složenost se obično odnosi na broj čvorova koji se generišu tokom pretraživanja, a **prostorna složenost** preko broja čvorova koji se čuvaju u memoriji.

4. Strategije za slepo (neinformisano) pretraživanje

- Ne postoji znanje o stanjima izvan definicije problema.
- Moguće je samo generisati potomke u stablu i utvrditi da li je neko stanje cilj ili nije.
- Pretraživanje se svodi na razlikovanje redosleda otvaranja čvorova i postoje 5 strategija:
    - 1) pretraživanje prvo u širinu
    - 2) uniform-cost search
    - 3) pretraživanje prvo u dubinu
    - 4) ograničeno pretraživanje u dubinu
    - 5) pretraživanje prvo u dubinu sa iterativnim povećanjem dubine

1) Svodi se na otvaranje svih čvorova na jednoj dubini pre otvaranja drugih. Otvara se prvo koren pa njegovi potomci, pa potomci njegovih potomaka, itd.

   Ova strategija je najoptimalnija ako je cena svake akcije ista. Vremenska i prostorna složenost nisu sjajne uopšte - O(b^d)

2) Otvara čvor sa najmanjom cenom putanje korišćenjem reda prioriteta za granične čvorove. Ukoliko svaki potez ima istu cenu, svodi se na pretraživanje prvo u širinu.
   Budući da je vođena cenom putanje a ne dubinom, teže je izraziti vremensku i prostornu složenost ove strategije ali ona definitivno **može biti mnogo gora od O(** b^d **)** ukoliko se primeni na ogromno stablo sa mnogo malih koraka.

3) Otvara se prvo najdublji čvor, čvor koji je najdalje korenu. Zaustavlja se kada čvor nema više potomaka. Poslednji pronađeni čvor se izbacuje iz skupa graničnih i vraća se na sledeći čvor sa prethodnog nivoa. Koristi Stek (LIFO) za skladištenje čvorova gde se poslednji čvor bira za otvaranje.
   Vremenska složenost može biti katastrofalna ukoliko se ciljni čvor nalazi u skroz drugom podstablu od početnog. Ukoliko je stablo pretraživanja beskonačno veliko, tolika je i vremenska složenost. Prostorna složenost je znatno manja od prethodnih strategija zbog toga što drži samo jednu putanju od korena do lista zajedno sa neotvorenim rođacima. Dakle ima zauzeće memorije od **O(bm)** čvorova.

4) Nedostaci prethodne strategije se rešavaju ograničenjem dubine pretrage gde se čvorovi na toj dubini tretiraju kao da nemaju decu. Nije optimalan ni kompletan u slučajevima da je ciljni čvor ispod ograničenja dubine pa je stoga samo koristan kada iz definicije problema možemo jasno da uočimo gde možemo da ograničimo dubinu pretrage. Vremenska složenost ovog algoritma je O(b^l), a prostorna složenost je O(b l).

5) U kombinaciji sa prethodnom strategijom, iterativno povećanje dubine dovodi do najboljeg limita pretraživanja. Limit na početku obično bude 0, pa se uveća na 1,2,... pa sve do cilja. Ovaj algoritam pretraživanja koristi prednosti pretraživanja prvo u dubinu i pretraživanja prvo u širinu. Prostorna složenost mu je stoga O(bd), dok vremensku složenost usled ponovnog generisanja čvorova na višim nivoima iznosi O(b^d).

5. Strategije za informisano (heurističko) pretraživanje

Predstavlja efikasniju vrstu pretraživanja u odnosu na neinformisanu budući da se pored same definicije problema koristi i znanje o samom problemu.

Selekcija čvora za otvaranje se vrši na osnovu **funkcije evaluacije f(n)** koja bira čvor sa najmanjom vrednošću troškova tj. s najmanjom vrednošću same funkcije.

Strategije informisanog pretraživanja se primarno razlikuju po izboru funkcije evaluacije. Većina koristi **heurističku funkciju h(n)** koja predstavlja procenjenu cenu najpovoljnije putanje od stanja u čvoru n do ciljnog stanja. **Ona je po prirodi nenegativna i opadajuća funkcija i u ciljnom stanju iznosi 0.**

1) Pohlepno pretraživanje prvo najbolje

Ovde je funkcija evaluacije = heuristička funkcija, tj. ovaj algoritam procenjuje čvor samo na osnovu vrednosti heurističke funkcije. Ovaj algoritam pokušava u svakom koraku da pohlepno dođe do cilja iako potencijalno može upasti u beskonačnu petlju. Nije kompletan. U najgorem slučaju prostorna složenost je O(b^m) ali se može značajno poboljšati sa dobrom heurističkom f-jom.

2) A * pretraživanje

Predstavlja jednu od poznatijih strategija informisanog pretraživanja. Ovde je f(n) = g(n) + h(n).
Predstavlja procenjenu cenu najpovoljnijeg rešenja koje prolazi kroz n. Da bi A* pretraživanje bilo optimalno i kompletno heuristička funkcija mora da zadovolji određene uslove, a to su **dopustivost** (dodeljuje vrednost manju ili jednaku stvarnoj ceni) **i monotonost** (monotono opadajuća).

Tree-search verzija A* pretraživanje je optimalna ako je h(n) dopustiva, a Graph-search verzija je optimalna ako je h(n) monotona.

Prostorna složenost ovog algoritma je prevelika tako da je on neupotrebljiv za mnoge kompleksne probleme.

6. Heurističke funkcije

Predstavlja procenjenu cenu najpovoljnije putanje od stanja u čvoru n do ciljnog stanja. **Ona je po prirodi nenegativna i opadajuća funkcija i u ciljnom stanju iznosi 0.**

Kvalitet heurističke funckije se najčešće definiše po **faktoru grananja. Dobro dizajnirane heurističke funkcije će imati efektivni faktor grananja blizu 1.**

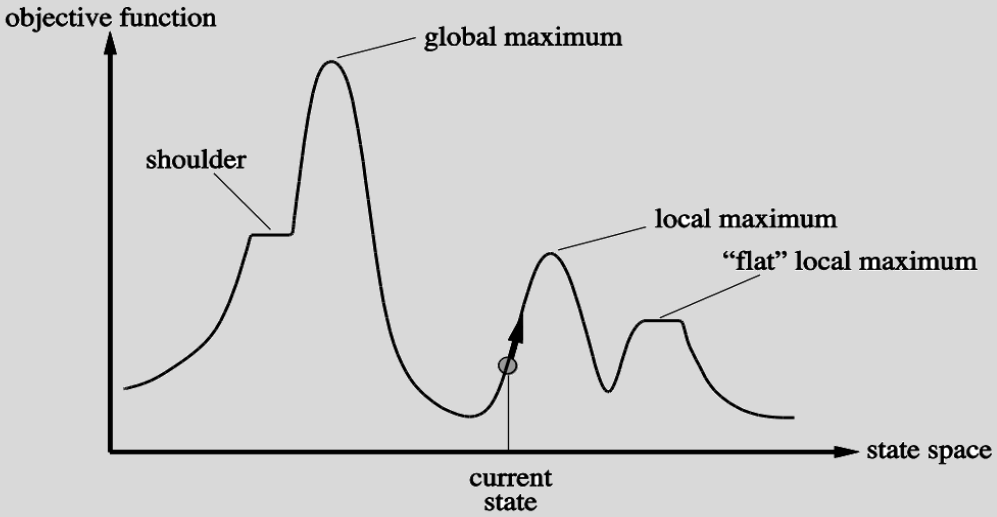
Heurističku f-ju možemo generisati na više načina, **podučavanje agenata primerima i pojednostavljenje problema, kao i grananje na podprobleme** su neki od njih.

7. Algoritmi za lokalno pretraživanje

Lokalno pretraživanje predstavlja proces procenjivanja i izmene jednog ili više trenutnih stanja. Ovi algoritmi pogodni su kod onih problema kod kojih nam je samo bitno rešenje (ciljno stanje) ali ne i putanja kojom se dolazi do tog stanja. Koristi “pejzaž”

Osnovni princip lokalnog pretraživanje je da se uzima tekući čvor i pomera se samo do njemu susednog čvora. Ovi algoritmi imaju dve dobre osobine (1) **Koriste vrlo malo memorije** i (2) **često nalaze prihvatljiva rešenja u velikim ili bekonačnim prostorima stanja na koja se ne mogu primeniti sistematični algoritmi za pretraživanje.**

Kompletan algoritam za lokalno pretraživanje uvek pronalazi rešenje ako ono postoji, a optimalan algoritam lokalnog pretraživanja uvek pronalazi globalni minimum/maksimum.



1) Algoritam penjanja uzbrdo

Ovaj algoritam sastoji se od petlje koja se pomera u pravcu većih vrednosti, odnosno uzbrdo. Završava se kada dostigne vrh, odnosno kada ni jedan sused nema veću vrednost.
Algoritam penjanja uzbrdo se često naziva i **gramzivo lokalno pretraživanje** jer on uvek "grabi" susedno stanje bez gledanja nekoliko poteza unapred. **Ovaj algoritam uvek vrlo brzo napreduje ka cilju.**

Međutim algoritam penjanja uzbrdo se obično zaglavi iz slededih razloga:

- lokalni maksimum, koji stvori iluziju dostignutog cilja
- lukovi
- ravnine (shoulder i "flat"), koje stvore iluziju nepromenjenog stanja u budućnosti.

Ovaj algoritam ima mnoštvo svojih varijanti, neke od njih su:

- **stohastičko penjanje uzbrdo** (unosi element verovatnoće u izbor putanja koje će potencijalno kasnije voditi uzbrdo iako možda privremeno krene “nizbrdo”).
- **random-restart penjanje uzbrdo** (sprovodi seriju algoritama penjanja uzbrdo iz proizvoljno izgenerisanog početnog stanja).

2) Lokalno pretraživanje u snopu

Prati **k proizvoljnih stanja** umesto jedno i ako je bilo koji od potomaka izgenerisanih cilj, algoritam se zaustavlja, a u suprotnom bira k najboljih stanja i nastavlja.

Razlikuje se od random-restart algoritma po tome što u random-restart algoritmu svaki proces pretraživanja se odvija nezavisno od ostalih, dok **u lokalnom pretraživanju u snopu, korisne informacije se prenose kroz paralelne procese**.

Problem može da nastane kada se odabranih k stanja ne razlikuju mnogo pa se **svede na pretraživanje uzbrdo**.

Taj problem se rešava jednom varijantom ove pretrage, **stohastičko lokalno pretraživanje u snopu** gde se **bira k slučajnih potomaka** pri čemu je verovatnoća da bude odabran jednaka njegovoj vrednosti.

3) Genetski algoritmi

Predstavljaju varijantu stohastičkog pretraživanja u snopu u kom se **potomci generišu kombinovanjem dva roditelja.** Počinje sa **populacijom** i svako stanje se ocenjuje **fitness funkcijom** koja vraća veće vrednosti za bolja stanja. Nakon toga se na osnovu te fitness funkcije i određene verovatnoće **biraju stanja za “reprodukciju”**. Sledeća generacije se **dodatno meša** gde

postoji određena verovatnoća da se parovi ukrste dok neki neće. Konačno, na kraju se vrši **proizvoljna mutacija sa malom neizvesnom verovatnoćom**.

Prednost im je u procesu ukrštanja roditelja zbog čega nekad može lakše da se “preskoči” ka potencijalnom cilju.

U praksi genetski algoritmi imaju velikog uticaja u domenu optimizacije, ali ostaje još posla u identifikaciji uslova pod kojim genetski algoritmi daju dobre rezultate.

8. Pretraživanje sa nedeterminističkim akcijama

Budući da se pretraživanje izvodi u nedeterminističkom okruženju (kao i mnogi problemi iz realnog sveta), rešenje problema se svodi na **plan za nepredvidljive situacije**. Ovakav plan govori šta treba uraditi u zavisnosti od zapažanja koja nam govore u kom stanju se nalazimo.

Za nedeterministička okruženja rešenja mogu sadržati if-else izraze, što znači da se ova rešenja mogu predstaviti kao **stabla umesto sekvence**. Takva stabla se nazivaju **AND-OR stabla pretraživanja**.

OR čvorovi su oni u kojima agent odlučuje šta će da uradi, a **AND čvorovi** su oni koji predstavljaju „odluku“ okruženja koja akcija će u stvari da se izvrši. Rešenje predstavlja podstablo koje mora da sadrži jednu granu iz svakog OR čvora i sve grane iz AND čvorova.

Za probleme u kojima akcije mogu da se ne izvrše pravilno(ne postoji aciklično rešenje), moguće je uvesti **petlje** koje ponavljaju neka stanja sve dok se ne dođe do željenog stanja.

9. Pretraživanje u okruženju koje se ne može potpuno sagledati

Ovde agent može samo da pretpostavi u kom stanju se trenutno nalazi i to se naziva **belief state**. Prepoznajemo tri vrste ovog pretraživanja:

1) Pretraživanje bez zapažanja

Agent može da se snađe za pojedine probleme u kojima agent “barata” samo sa verovatnim stanjima. Primer toga bi bio lekar koji prepisuje pacijentu antibiotik pre nego što odradi seriju skupih i dugih testova gde u velikoj većini slučajeva, na osnovu tih simptoma, prepisani antibiotik rešava problem bez tih testova.

Kod agenata koji nemaju senzore, umesto grafa realnih stanja, pretražuje se **graf "verovatnih" stanja.** Rešavanje ovakvih problema se svodi na **analiziranje realnih stanja i rešavanje istih jedno po jedno**. Kada se nađe rešenje za jedno realno stanje proverava se da li je to rešenje i za ostala, u slučaju da nije **vraća se na prethodno stanje i traži se sledeće rešenje**.

2) Pretraživanje sa parcijalnim zapažanjem

Za formalnu specifikaciju sistema, gde agent može parcijalno da sagleda okruženje, koristi se funckija **PERCEPT koja vraća sva zapažanja agenta u datom stanju**. U okruženjima koja se ne mogu sagledati ova funkcija daje null, dok u okruženjima koja mogu potpuno da se sagledaju ova funkcija predstavlja čitavo stanje.

Problem kod parcijalnog zapažanja može da nastane kada **više različitih stanja proizvedu ista zapažanja usled preklapanja**. Rešenje za ove probleme predstavljaju AND-OR stabla kao i contigency plan, tj. plan za nepredviđene situacije. Zato agent sprovodi neki od algoritama pretrage koji mu vraća rešenje problema u vidu plana za nepredviđene situacije koji se sastoji od if-else izraza.

Agent testira if-else uslove i na osnovu toga izvršava određene akcije. **Agent mora da ažurira verovatna stanja u zavisnosti od zapažanja** koristeći funkciju UPDATE.

3) Online pretraživanje u nepoznatom okruženju

Agenti u offline pretraživanju računaju rešenje pre nego što zakorače u stvaran svet. Online pretraživanje **kombinuje pretraživanje sa konkretnim akcijama**. Koristi se u dinamičnom i poludinamičnom okruženju gde agent dobija penale ako računa previše dugo kao i u nedeterminističkim okruženjima.

Agenti se ovde najčešće susreću sa **problemom istraživanja okruženja** gde koristi sve svoje akcije kao eksperimente u korist prikupljanja informacija o okruženju. Agent ništa ne računa unapred **već rešava problem akcijama kojim dobija zapažanja**. Koristeći ta zapažanja agent stvara **mapu** na osnovu koje odlučuje kako će ići dalje. Ovde se koriste najčešće algoritmi pretrage **prvo u dubinu**, kao i algoritam **penjanja uzbrdo sa memorijom**.

10. Pretraživanje u igrama. MINIMAX stablo pretraživanja. Optimalno odlučivanje u igrama sa dva i više igrača.

U igrama se često javlja **okruženje sa više agenata** u kom agenti moraju da razmatraju akcije drugih agenata i uticaj njihovih akcija. Najčešće su to **kompetitivna okruženja** u kojima su ciljevi agenata u konfliktu (adversarial search).

Obično su u pitanju igre u kojima svaki igrač igra naizmenično dok jedan igrač ne pobedi ili dok ne bude nerešeno. Agenti najčešće imaju mali skup mogućih akcija. **Igre u suštini predstavljaju determinističko, sagledljivo okruženje.**

Igra se može formalno opisati kao problem pretraživanja preko slededih elemenata:

- S_0 **- početno stanje**
- Player(s) - koji igrač je na potezu u stanju s
- **Actions(s) - skup legalnih poteza u stanju s**
- **Result(s,a) - tranzicioni model koji definiše rezultat poteza**
- Utility(s,p) - funkcija cilja, daje konačan rezultat u završnom stanju s za igrača p, na primer u šahu rezultat ove funkcije može biti pobeda, poraz ili nerešeno, sa vrednostima +1,0,1/2

Početno stanje, akcije i tranzicioni model definišu stablo pretraživanja u igrama u kome su čvorovi stanja u igri a grane su potezi.

MAX igrač mora da nađe strategiju koja određuje njegov potez u početnom stanju, zatim odgovor na svaki mogući potez MIN igrača i tako dalje

MINIMAX algoritam koristi rekurziju za kreiranje MINIMAX stabla pretraživanja.

MINIMAX algoritam funkcioniše tako što **listovi dobiju vrednost funkcije cilja**. **MIN čvorovi uzimaju najmanju vrednost svojih potomaka, a MAX čvorovi najveću vrednost svojih potomaka.** I to se ponavlja za sve MIN i MAX čvorove. **U korenu(MAX čvor) identifikujemo odluku minimax algoritma**, a to je akcija koja nas vodi u stanje koje ima maksimalnu vrednost.

MINIMAX algoritam je sličan pretraživanju prvo u dubinu. **Vremenska složenost algoritma je O(bm), a prostorna složenost O(bm)** za verziju algoritma koja generiše sve akcije odjednom, odnosno O(m) za verziju koja generiše jednu po jednu akciju. **Za realne igre, vremenska složenost je apsolutno nepraktična**, ali ovaj algoritam je osnova za matematičku analizu igara i za druge praktičnije algoritme.

Kada su u pitanju igre sa više od dva igrača, potrebno je uvesti vektor koji se vezuje za svaki čvor. U završnom stanju vektor sadrži vrednosti f-je cilja sa tačke gledišta svakog igrača.

11. Alfa-beta odsecanje kod MINIMAX algoritma.MINIMAX do određene dubine.

Budući da skoro polovina stanja mogu da se odstrane iz stabla jer ne utiču na krajnji rezultat koristi se algoritam koji se zove **alfa-beta odsecanje**. **Kada se primeni na MINIMAX stablo pretraživanja vratiće isti potez kao rezultat ali sa znatno manjim brojem grana.**

Alfa-beta algoritam se može primeniti na bilo koju dubinu i njime se najčešće odsecaju cela podstabla, a ne samo listovi. Generalni princip je sledeći: posmatramo neki čvor n u stablu pretraživanja tako da igrač ima opciju da izabere taj čvor. Ako igrač ima neku bolju opciju m bilo u čvoru roditelju od n bilo u nekom plićem čvoru tada se čvor n nikada nede dostići u igri. Znači da kada saznamo dovoljno o n (otvaranjem određenog broja potomaka) da ovo zaključimo možemo ga odseći.

Alfa i beta predstavljaju *trenutne* najbolje vrednosti za MAKS, tj. za MIN čvorove. Čvorovi se tokom pretrage seku **redom** dokle god se pronalazi gora vrednost od trenutne alfa i bete. Budući da se seku redom **efikasnost zavisi od reda otvaranja čvorova**.

Iako korisna, i alfa-beta verzija algoritma m**ora da pretraži do završnog stanja makar za deo stabla**. Ova dubina je obično nepraktična, jer potez treba da se odigra u razumnom vremenu. Umesto uopštenog MINIMAX algoritma u praksi se najčešde primenjuje **MINIMAX do neke zadate dubine na kojoj se izračunava heuristička funkcija(EVAL) evaluacije za listove koji nisu završna stanja.**

Metod pretraživanja do određene dubine ima određene nedostatke, a glavni je taj da neka obećavajuća putanja na određenoj dubini kasnije vodi u lošu situaciju. Ovo se drugačije naziva i **efekat horizonta.** To može donekle da se prevaziđe ako se u toku igre **menja dubina pretraživanja.**

F-ja evaluacije daje **procenu očekivanog rezultata u igri iz *trenutne* pozicije.** Stoga **uspešnost programa koji igra u velikoj meri zavisi od kvaliteta f-je evaluacije.** Obično f-ja evaluacije može matematički da se izrazi kao **linearna f-ja koja određenim potezima dodaje niže ili više vrednosti uz neki koeficijent koji dodatno kvalitativno određuje potez ili neki element poteza.**

12. Logička posledica i pravila izvođenja. Predstvaljanje znanja u iskaznoj logici. Logičke ekvivalencije, valjane i zadovoljive formule.

Logika predstavlja formalni jezik za opis informacija tako da se iz tog opisa mogu izvući neki zaključci i ona predstavlja osnovu za izgradnju baze znanja. **Baza znanja** predstavlja jedan centralizovan skup iskaza koji predstavljaju neke informacije o svetu oko nas. Iskazi su napisani u nekoj **sintaksi.** Uz nju koristimo i **semantiku** pomoću koje iskazima dodeljujemo vrednosti (true/false). **Interpretacija** predstavlja dodelu vrednosti true ili false iskazima.

Kažemo da je iskaz α **logička posledica** baze znanja KB ako i samo ako u svakoj interpretaciji u kojoj su svi iskazi u KB tačni sledi da je i α tačno. Ovo obeležavamo sa KB $\models$ α.

Da bismo na osnovu neke baze znanja došli do logičke posledice potrebno nam je neko logičko rezonovanje koje iziskuje **pravila izvođenja.** Za pravilo izvođenja i možemo reći KB $\vdash_i$ α, što znači da se iskaz α može izvesti iz KB korišćenjem pravila izvođenja i.

Pravilo izvođenja je **saglasno tj. neprotivurečno** ako KB $\vdash_i$ α, onda važi i KB $\models$ α što znači da se tim pravilom izvođenja **izvode *samo* logičke posledice**.

Takođe, može biti **kompletno** ako KB $\models$ α , tada KB $\vdash_i$ α, odnosno pravilom izvođenja *i* se mogu **izvesti sve logičke posledice**.

Sintaksu iskazne logike čine iskazni simboli i logički veznici ($\neg$, $\wedge$, $\vee$, $\Rightarrow$, $\Leftrightarrow$). Iskazne formule su iskazi i iskazi kombinovani sa logičkim veznicima.

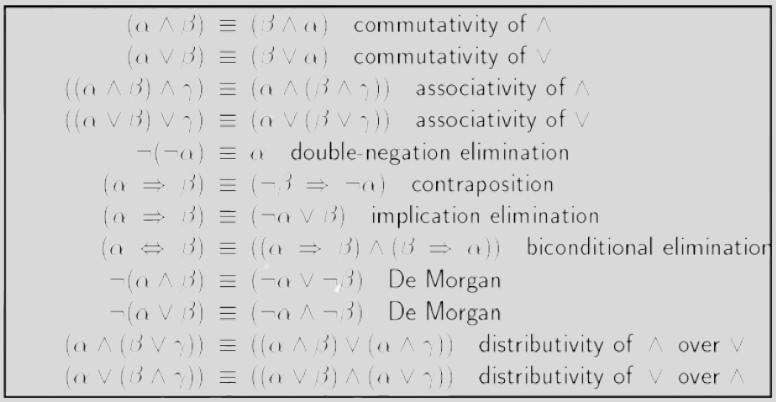
Iskazna formula je **valjana ako je tačna u svakoj interpretaciji.** Valjane formule zovemo još i tautologije. Primer jedne valjane formule je: $P \vee \neg P$

Dedukcijom možemo da pokažemo da je α $\models$ β tako što pokažemo da α $\Rightarrow$ β

Formula je **zadovoljiva** ako postoji interpretacija u kojoj je ona tačna.

Dve logičke formule su ekvivalentne, ako su tačne u istim interpretacijama.

Neke od najpoznatijih logičkih ekvivalencija:



Pravilom izvođenja možemo proveriti da li je neka formula logička posledica nekog skupa formula. Služi za konstruisanje novih formula koristeći skup postojećih. Najpoznatije je Modus Ponens.

Pravilo rezolucije je jedna varijacija pravila izvođenja koja je **kompletna**. Uvodi **literale** (iskaz ili njegova negacija) i **klauzulu** (disjunkcija literala). Literali koji negiraju jedan drugog se nazivaju **komplementarni literali**. Pravilo rezolucije iziskuje da se baza znanja i upiti sastoje iz klauzula. Svaka formula iskazne logike se može napisati kao konjukcija klauzula, i za sve tako napisane formule kažemo da pripadaju **konjuktivnoj normalnoj formi (KNF).**

Postoji procedura kojom se formule prevode u KNF:

1. Eleminišemo ekvivalenciju zamenjujući je sa dve implikacije
2. Eliminišemo implikaciju zamenjujući je sa ¬x ˅ y
3. KNF zahteva da negacije bude uz literal, što postižemo korišdenjem demorganovih pravila i elminicijom duple negacije.
4. treba primeniti pravilo distributivnosti operacije V nad operacijom ^ i dobija se formula koja je u konjuktivnoj normalnoj formi.

13. Sintaksa i semantika logike prvog reda. Simboli i interpretacije. Kvantifikatori. Baza znanja sa logikom prvog reda

Budući da iskazni račun nije dovoljno moćan uvodi se logika prvog reda. Logika prvog reda je zapravo predikatski račun prvog reda.

Sintaksu logike prvog reda čine simboli koji označavaju objekte, relacije i funkcije.

Ti simboli su:
- konstante (koje predstavlju objekte),
- predikatski simboli (relacije) i
- funkcijski simboli (funkcije)

Term je logički izraz koji opisuje neki objekat. U terme spadaju konstante, promenljive i funkcije.

Atomske formule (atomi) su predikati koji sadrže listu terma kao argumente u zagradi. Kompleksne formule nastaju kombinacijom atomskih formula i iskazne logike.

Kvantifikatori su elementi predikatske logike koji se koriste kada se opisuju osobine cele kolekcije objekata. Imamo ih dva, ∀ **- (für alle, tj. za sve)** ∃ **- (egsistiert, tj. postoji)**

Primer: ∃ x Kruna(x) ^ NaGlavi(x, Dzon)

Rečenice se dodaju u baznu znanja koristeći “**Tell**”, na primer: Tell(KB, Kralj(Džon))

Analogno tome, bazi znanja možemo da postavimo pitanje sa “**Ask**”, na primer:
Ask(KB, Kralj(Džon)).

Aksiomama se predstavljaju neke osnovne informacije iz kojih se mogu izvući **teoreme.**

Domen sadrži samo element koji su navedeni kao konstante. Primer domena bi bila jedna kraljevska porodica gde su objekti u domenu ljudi.

Rezime

Korišćenje logike prvog reda

- reprezentacija znanja
  - domen
- aksiome, definicije i teoreme
- primeri domena: kraljevska porodica, brojevi, skupovi, liste, vumpus svet
- infiksna i prefiksna notacija

Sintaksa i semantika logike prvog reda

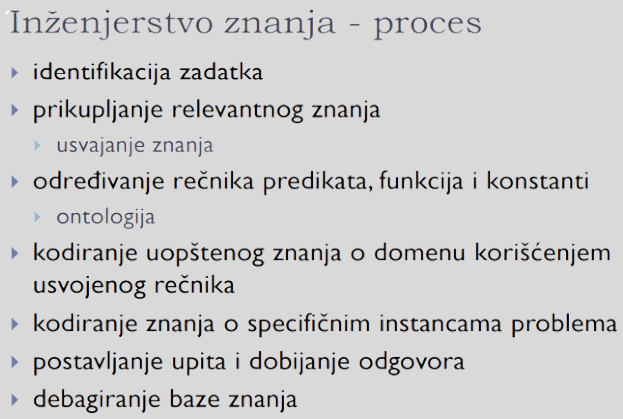
- simboli kojima se predstavljaju objekti, relacije i funkcije
- vrste simbola: konstante, predikati (relacije) i funkcije
- arnost relacije i funkcija
- interpretacija
- termi
- atomske rečenice
- složene rečenice
- kvantifikatori

14. Koraci razvoja projekta u oblasti inženjerstva znanja.

Inženjerstvo znanja je proces konstruisanja baze znanja. Postoje specifične baze znanja u kojima su svi upiti unapred poznati i čiji je domen ograničen. To su baze **znanja za specifične svrhe** (special-purpose). Pored njih postoje i general-purpose baze znanja, tj generalne baze znanja i one sadrže široki skup ljudskog znanja.

Projekti inženjerstva znanja mogu znatno da se razlikuju ali se svaki sadrži od:

- Identifikacija zadatka
  - Identifikacija svih pitanja na koje baza znanja treba da dâ odgovor.
  - Identifikacija vrsta činjenica koje će biti dostupne u različitim instancama problema
- Prikupljanje relevantnog znanja (usvajanje znanja)
  - Razgovor sa ekspertima na temu
- **Određivanje rečnika predikata, funkcija i konstanti (Potencijalno najkritičniji deo)**
  - Prevođenje važnih koncepata domena u logičke koncepte.
  - **Rezultat je rečnik, tj. ontologija domena**
- Kodiranje uopštenog znanja o domenu korišdenjem usvojene ontologije
  - Pišu se aksiome za sve koncepte ontologije
  - Uočavanje i ispravka potencijalnih nedostataka u ontologiji domena
- Kodiranje znanja o specifičnim instancama problema
  - Pisanje jednostavnih atomskih rečenica o instancama koncepata
- Postavljanje upita i dobijanje odgovora
  - Pozivanje procedura zaključivanja nad aksiomama i činjenicama baze znanja radi dobijanja novih činjenica i teorema
- Debagiranje baze znanja
  - Testiranje baze znanja postavljanjem upita i analizom odgovora da bi se utvrdile eventualne greške



15. Zaključivanje u logici prog reda.

Zaključivanje u logici prvog reda je osnova za sve moderne logičke sisteme.

Nastalo je u cilju kreiranja pravila koje će pomoći računaru da stekne sposobnost zaključivanja nekih očitih problema.

Proces zaključivanja koji zahteva pronalaženje zamena kojima se različiti logički izrazi mogu napraviti identičnim se naziva **unifikacija**. Predstavlja glavni koncept rezonovanja u logici prvog reda.

Unifikacija je **algoritam za određivanje zamena (supstitucija)** koje treba izvršiti da bi se uparila dva izraza predikatskog računa. Zamena ili supstitucija odnosi se na promenu formule tako što se **promenljive koje su uz univerzalne kvantifikatore zamenjuju sa odgovarajudim termom.**

Unifikacija uzima dve formule pred. računa i vraća njihov unifikator ukoliko on postoji. Da bi ta f-ja radila, potrebno je da eliminišemo promenljive koje su vezane za egzistencijalne kvantifikatore (∃) i taj proces se zove **proces skolemizacije**.

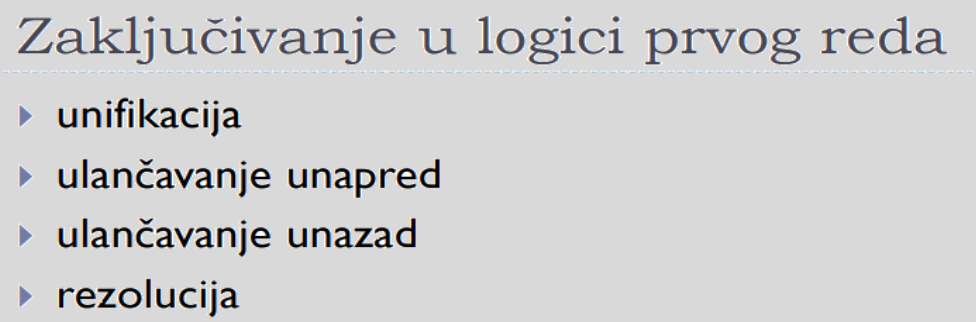
Unifikatori mogu da se primene i na baze znanja. Možemo imati više od jednog ali uvek je jedan najopštiji, tj. najmanje restrikcija postavlja.

Postoje dva tipska postupka zaključivanja u logici prvog reda koristeći unifikaciju:

- **ulančavanje unapred** (forward-chaining) - rešavanje problema počinje od poznatih činjenica
- **ulančavanje unazad** (backward-chaining) - prvo se uoči cilj koji treba da se dostigne, pa pravila koja vode do cilja. Određuju se uslovi koji moraju biti ispunjeni da bi se primenom tih pravila došlo do cilja.

Ova dva postupka mogu da se primene na većinu baza znanja, ali ne i na sve.

Metoda rezolucije može da se proširi u oblast logike prvog reda.(“Rezolucija u logici prog reda” str. 83/130)



16. Ontološki inženjering. Kategorije i objekti. Sistemi rasuđivanja za kategorije

Predstavljanje svega što postoji je neograničen problem, umesto toga ideja ontološkog inženjeringa je da **napravi okvirnu ontologiju u koju će moći da se naknadno ugrade znanja o specifičnim elementima**. Na primer, možemo definisati šta je to fizički objekat a detalji o konkretnim objektima (televizor, robot,...) mogu se naknadno dobiti. Takava generalna ontologija se naziva **gornja ontologija.**

Postoje dve vrste ontologija, to su **ontologije za opšte namene** i **ontologije za specifične namene.** Uopštena ontologija se može primeniti na više ili manje svaki specifičan domen, dok su ontologije za specifične namene specifične za konkretan domen. Većina iole kompleksnih VI aplikacija koriste ontologije za specifične namene.

Organizacija objekata u kategorije je jedna od glavnih koncepata reprezentacija znanja.

Veliki deo rasuđivanja se obavlja na nivou kategorija preko kojih se takođe mogu odrediti osobine objekata. Na primer, ako agent utvrdi na osnovu narandžaste hrapave kore, okruglog oblika da se radi o pomorandži on zaključuje da se taj objekat može ubaciti u voćnu salatu.

Odnos podkategorije uređuje kategorije u tzv. taksonomije.

Postoje dva načina da se u logici prvog reda predstave kategorije, to su objekti i predikati.

Kategorije se koriste da bi se baza znanja mogla bolje organizovati i pojednostaviti korišdenjem koncepta **nasleđivanja**.

Postoje dva sistema za rasuđivanje nad kategorijama:

- **semantičke mreže,**
- **deskriptivne logike**

Semantičke mreže **definišu grafičku notaciju** kao pomoć vizualizicji baze znanja, dok deskriptivne logike **obezbeđuju formalni jezik za konstruisanje** i kombinovanje definicija kategorija.

Definišu efikasne algoritme za **izvođenje zaključaka pripadnosti, tj. određivanja** odnosa pod i nad kategorije.

Rezime

Kategorije i objekti

- organizacija objekata u kategorije
- rezonovanje na nivou kategorije
- logika prvog reda: objekti, predikati
  - KosarkaskaLopta(k)
  - KosarkaskaLopta, Clan(k, KosarkaskaLopta)
- podkategorija
  - Poskup(KosarkaskaLopta,Lopta)
- nasleđivanje
  - bolja organizacija i pojednostavljenje baze znanja
  - primer: Hrana, Voće, Jabuke
- taksonomija

17. Izražavanje nesigurnosti. Osnove teorije verovatnoće

Nesigurnosti se često javljaju u rešavanju problema realnog sveta budući da se često javljaju nedeterministička okruženja koja nisu potpuno sagledljiva.

Agenti za rešavanje problema pretraživanjem i logički agenti su savladavali nesigurnost predstavljanjem svih mogućih stanja u kojima se agent može naći i generisanjem plana akcija. Međutim, taj pristup ima određene nedostatke:

- Agent mora da sagleda sve moguće varijante stanja, što dovodi do jako velikih i kompleksnih reprezentacija znanja.
- Ponekad ne može uopšte doći do rešenja a agent svakako mora izvršiti određenu akciju. Ovo bi se potencijalno rešilo uvođenjem ocene osobine mogućih rešenja.

Korišćenjem logike za rešavanje problema kompleksne prirode bi dovelo do ogromnog broja uzroka i posledica. Takođe, nekad priroda problema nalaže da ne postoji teorija iza problema, a ponekad ne postoji ni dovoljno praktičnog znanja o istom.

Primer toga je problem zubobolje. Pacijent dolazi kod stomatologa sa zuboboljom. Ako pacijent ima zubobolju, ne mora da znači da ima karijes, uzročnik može da bude mnoštvo drugih problema koji kao simptom imaju zubobolju. Analogno tome, ako pacijent ima karijes, ne mora da znači da će imati zubobolju. Jednostavno takav problem je jako teško predstaviti logikom prvog reda bez korišćenja bilo kakve verovatnoće.

Osnovna alatka za obradu “stepena verovanja” je teorija verovatnoće

Agenti koji koriste verovatnoću mogu da odrede stepen verovatnoće da je nešto tačno (između 0 i 1). Primer toga bi bio verovatnoća da neko ko ima zubobolju ima i karijes je 80%, tj. 0,8.

Pojedinačni ishodi nekog događaja se nazivaju elementarni događaji.

Osnovni aksiom teorije verovatnoće kaže da svaki elementarni događaj ima verovatnoću veću ili jednaku od 0 i manju ili jednaku od 1 i da je zbir verovatnoća svih elementarnih događaja iz skupa Ω jednak 1 (100%). (skup Ω je skup svih elementarnih događaja $\omega 1$, … ωi).

Verovatnoća izražena sa P(A) se naziva bezuslovna verovatnoća i predstavlja verovatnoću da je nešto tačno u odsustvu bilo kojih dodatnih informacija.

Pored bezuslovne verovatnoće postoji i **uslovna verovatnoća** koja se označava sa P(A|B) koja definiše **verovatnoća da je tačno A uz uslov da je ispunjeno B**.

Uslovna verovatnoća može da se izrazi iz bezuslovne na sledeći način: $P(A|B) = \frac{P(A \wedge B)}{P(B)}$

18. Bajesovo pravilo i Bajesova mreža.

Bajesovo pravilo predstavlja osnovu za skoro sve moderne VI sisteme za rezonovanje u prisustvu nesigurnosti. **Ono nam omogućava da izračunamo uslovnu verovatnoću čak iako ne znamo P(A ∧ B).**

$$P(B|A) = \frac{P(A|B)P(B)}{P(A)}$$

Bajesovo pravilo je važno budući da često raspolažemo sa klauzalnim znanjima P(simptom|bolest) , a želimo da izvedemo rasuđivanje na osnovu nekih posledica ili dokaza P(bolest|simptom).

Alakta koja nam dodatne pomaže u tome je pojam **relativne verovatnoće**.

Ako znamo da je P(A|B1) = P(A|B2), tada se iz Bajesovog pravila može izvesti:

Kako nam znanje iz B1 i B2 daju isto znanje o A, znanje o A ne menja relativnu verovatnoću za B1 i B2.

$$\frac{P(B_1|A)}{P(B_2|A)} = \frac{P(B_1)}{P(B_2)}$$

Bajesova mreža predstavlja grafičku notaciju za predstavljanje uslovnih verovatnoća.

Bajesovu mrežu sačinjava skup čvorova (*usmereni aciklični graf*) **gde se za svaki čvor navodi uslovna verovatnoća P(čvorN | roditelj(čvorN)). Svakom čvoru se pridružuje tablica uslovnih verovatnoća. Za čvorove koji nemaju decu se uvodi bezuslovna verovatnoća.**

Verovatnoće sumiraju potencijalno beskonačan skup okolnosti u kojima se nešto može dogoditi.

Naziva se još i mreža uverenja, mreža verovatnoća, mreža zavisnosti,mapa znanja

19. Verovatnoća u nedeterminističkom okruženju. Očekivana korisnost. Funkcija korisnosti sa više atributa.

Teorija odlučivanja se bavi izborom akcije iz nekog skupa mogućih akcija na osnovu njihovih neposrednih ishoda. Result(a) predstavlja proizvoljnu promenljivu čije su vrednosti stanja koja su mogući ishodi akcije a. Verovatnoća ishoda s', uz zapažanja iz okruženja e označavamo na slededi način: $P(Result(a) = s' | a, \boldsymbol{e})$

Agentove preferencije se izražavaju funkcijom korisnosti U (s) i daje numeričku vrednost poželjnosti stanja.

Očekivana korisnost akcije uz zapažanje predstavlja se sa f-jom EU(a | e)

Agent treba da izabere akciju koja maksimizira njegovu očekivanu korisnost, i to se naziva princip maksimalne očekivane korisnosti.

Funkcija korisnosti za raličite akcije može da se poredi, U (a) > U (b) naznačava da preferiramo ishod a u odnosu na b. Možemo imati i U (a) < U(b) i U (a) = U (b).

Funkcija korisnosti nekad treba da **ima više atributa** da bi “imala smisla”. Recimo da imamo šansu da uzmemo zagarantovanih milion dolara kao prvu opciju, a kao drugu opciju možemo da rizikujemo tih milion (bacamo novčić npr.) dolara i da uzmemo 2.5 miliona dolara. U zavisnosti od toga da li je “vredno” tih početnih milion dolara treba razmotriti da li je vredno kockati se. Funckija korisnosti bi tu rekla da treba rizikovati za tih dodatnih 1.5 milion dolara dok ljudski razum gotovo uvek ne bi. Kada ima vise opcija traži se **dominantija opcija**, tj. opcija koja je **bolja po svim atributima od neke druge**. Ta pojava se takođe zove i **striktna dominacjia.** U slučaju kada ne znamo sve atribute, onda poredimo sve moguće atribute, i biramo slučaj gde su **svi mogući atributi bolje od druge opcije**, to se zove **stohastička dominacija**.

20. Mreže odlučivanja. Ekspertni sistemi zasnovani na teoriji odlučivanja.

U najopštijem obliku mreže odlučivanja predstvaljaju informacije o trenutnom stanju agenta, njegovim mogućim akcijama, stanju koje de biti rezultat izvršavanja akcija i korisnost stanja.

Sastoje se od 3 osnovna tipa čvora:

- **Čvorovi promjenljivih** - za svaki čvor promjenljivih **vezuje se uslovna verovatnoća** na osnovu verovatnoće roditelja (kao kod Bajesovih mreža).

- **Čvorovi odluke** - predstavljaju tačke gde onaj koji donosi odluku ima izbor akcija. **Kada se čvoru odluke dodeli neka vrednost on dobija ulogu čvora promenljive**.

- **Čvorovi korisnost**i - **predstavljaju funkciju korisnosti agenta**, za njih se vezuje opis agentove korisnosti kao funkcija čije su promljive roditelji čvora.

Procena mreže odlučivanja se vrši tako što se postavlja vrednost čvora za svaki čvor odlučivanja. Nakon toga se računaju sve uslovne verovatnoće za roditelj čvorove korisnosti nakon čega se **računa rezultujuća korisnost**. **Rezultat toga je akcija sa najvećom vrednošću f-je korisnosti.**

U mrežama odlučivanja, roditeljski čvor može da bude čvor promenljivih ili čvor odluke.

Inženjerstvo znanja za ekspertne sisteme koji uključuju teoriju odlučivanja se sastoji 6 komponenti:

1) Kreiranje modela uzroka i posledica. Predstavlja ucrtavanje veza između elemenata. Moguće je konsultovanje s eskpertima o temi u pitanju.

2) Pojednostavljenje modela i kreiranje kvalitativnog modela odlučivanja. Pregled prethodnog koraka i potencijalno stapanje stavki u jednu ili njihovo raščlanjivanje.

3) Određivanje verovatnoća. Podaci o verovatnoći se mogu dobiti na osnovu analize literature, baza podataka, ili subjektivne procene eksperta.

4) Određivanje korisnosti. Napravi se skala od 0 do 1 gde se svi mogući ishodi odrede na njoj.

5) Verifikacija i usavršavanje modela. Stručnjacima iz oblasti se predloži primer problema i poredi se odluka stručnjaka i odluka sistema. Rade se korekcije sistema po potrebi.

6) **Senzitivna analiza.** Važan korak u kom se proverava osetljivost odluka. Ukoliko male promene dovode do velikih razlika, onda treba nešto korigovati. Drugim rečima proverava se robusnost sistema. Najčešće nam oktriva da mnogi brojevi moraju da se odrede samo otprilike.

21. Pojam mašinskog učenja. Vrste mašinskog učenja. Nadgledano učenje i hipoteze

Kažemo da agent uči **ako on unapređuje svoj učinak** u rešavanju bududih zadataka nakon opservacije sveta koji ga okružuje.

Mašinsko učenje se odnosi na izgradnju prilagodljivih računarskih sistema čija je jedna od osnovnih uloga poboljšanje performansi. Mašinsko učenje ima veliku praktičnu primenu (*ALVINN - Autonomous Land Vehicle In a Neural Network, TD-Gammon, Sistemi za prepoznavanje govora, Sistemi za prepoznavanje rukom pisanog teksta*).

Na osnovu povratnih informacija sa kojima agent raspolaže razlikujemo dve vrste učenja:

- **nenadgledano učenje**
- **nadgledano učenje**

Nenadgledano učenje podrazumeva pronalaženje paterna u ulaznim podacima u okruženju gde ne postoji povratna informacija. Najčešći oblik je **klasterovanje(grupisanje) , objekti se grupišu u klastere u koijma su slični jedni drugima po nekoj osobini u odnosu na druge klastere.** Podseća na disjunktne klauzule.

Nadgledano učenje podrazumeva neku "obuku" koja se sastoji od N parova ulaz-izlaz (x1,y1), (x2,y2),...(xN,yN), gde svako y_i dobijamo nekom nepoznatom f-jom y = f(x). **Zadatak učenja je da otkrije hipotezu, tj. funkciju h koja predstavlja aproksimaciju f-je f.** Traži se hipoteza koja najbolje vrši **generalizaciju** i koja je **najkonzistentnija**. Da bi se to postiglo, potrebno je praviti određene kompromise u odabiru hipoteze.

22. Klasifikacija: metoda zasnovana na instancama, učenje stabla odlučivanja.

Za konačan skup vrednosti y, problem učenja nazivamo **klasifikacija**. Predstavlja razvrstavanje nepoznatih instanci u kategorije. Kada imamo skup od 2 elementa za y, onda je to **logička ili binarna klasifikacija**, a kada je to samo neki broj (jedna vrednost) - **regresija**.

Problem klasifikacije je određivanje kojoj klasi pripada instanca na osnovu vrednosti atributa. Postoji veliki broj metoda za rešavanje tog problema, neke od njih su:

- metoda zasnovana na instancama
- učenje stabla odlučivanja
- metoda potpornih vektora
- metode bajesove klasifikacije

Metoda zasnovana na instacama

Ovde se ne gradi eksplicitni model podataka u vidu neke funkcije kao što to radi većina metoda mašinskog učenja**. Klasifikacija se vrši na osnovu izabranog skupa primera za obuku**. Dva poznata koncepta koriste ovaj metod, **n-najbližih suseda i n-grami**.

N-najbližih suseda zasniva se na principu da se nepoznata instanca klasifikuje u klasu čije su instance najsličnije nepoznatoj koristeći relevantnu f-ju rastojanja (manje rastojanje, veća sličnost).

N-grami se koriste za rešavanje problema učenja koje karakterišu podaci nenumeričke prirode koristeći N-gramske profile. Ti profili predstavljaju reprezentaciju pogodnu za metode klasifikacije i koriste se kombinovano sa prethodnim konceptom.

Učenje stabla odlučivanja su jedan od najjednostavnijih, ali i jedan od najuspešnijih formi mašinskog učenja. Stablo odlučivanja kao f-ju **uzima vektor vrednosti atributa a kao rezutlat daje odluku**. Ono donosi odluku na osnovu serije testova. **Čvor koji nije list je jedan test, grane koje izlaze iz čvorova su mogući rezultati testa, a list je izlazna vrednost.**

Kako bi se napravilo najmanje moguće stablo koje je konzistentno koristi se algoritam Decision Tree algoritam koji je gramzive prirode. Koristi “podeli pa vladaj” strategiju koja predviđa da se **uvek prvo testira najznačajniji atribut (onaj koji najviše utiče na klasifikaciju, tj. na izlaz).** Probleme deli na manje potprobleme gde se svaki posmatraju kao novi problemi učenja i rešavaju se rekurzivno.

23. N-gram modeli prirodnog jezika. Klasifikacija teksta

Postoje dva osnovna razloga zašto želimo da osposobimo agenta da razume prirodni jezik, a to su komunikacija sa čovekom i drugi je usvajanje znanja .

Prirodnim jezikom smatramo jezik kojim govore i pišu ljudi u svakodnevnoj komunikaciji. Za razliku od formalnih, prirodni jezik se ne može u potpunosti opisati gramatičkim pravilima. Prirodni jezik sa sobom donosi probleme dvosmislenosti i veličine.

N-gram model se primenjuje **nad karakterima, nad sekvencama reči i nad slogovima** u cilju rešavanja pomenutih problema. Nad karakterima se implementira verovatnoća pojave određene sekvence N karaktera. Najčešće se koristi u ispravljanju grešaka nastalih kucanjem. Nad sekvencama reči se takođe koristi gde je rečnik koji se posmatra znatno veći. Druga bitna razlika je u tome što se taj rečnik povećava budući da se uvode nove reči.

Klasifikacija teksta predstavlja proces određivanja kojoj klasi pripada neki dati tekst. Moguće klasifikacije su identifikacija jezika, određivanje žanra, klasifikacija neke recenzije za film, sentimentalna analiza, detekcija spamova i sl.

Detekcija spamova je proces klasifikacije e-mail poruka kao spam ili ne-spam(ham). Predstavlja problem nadgledanog učenja.

24. Pribavljanje informacija. Ekstrakcija informacija.

Pribavljanje informacija (Information retrieval-IR) je **zadatak pronalaženja dokumenata koji sadrže informacje potrebne korisniku**. Najpoznatiji IR sistemi su Web pretraživači (na primer Google). IR sestemi se mogu okarakterisati slededim osobinama:

- **Korpus dokumenata** - Određivanje oblika dokumenta: paragraf, stranica, tekst na više str.
- **Upiti i upitni jezik** - Upitom korisnik definiše koje info. su mu potrebne. Upitni jezik može da ima fraze, listu reči, logičke operatore, nelogičke operatore
- **Skup pogodaka** - Podskup dokumenata koje IR sistem „smatra“ relevantnim za upit.
- **Prezentacija skupa pogodaka** - Rangirana lista dokumenata ili neki grafički prikaz rezultata

IR scoring funkcija(BM25) uzima dokument i upit i vrada numeričku vrednost (skor) koja je veća za relevantnija dokumenta. Tri faktora utiču na određivanje skora: **frekvencija pojavljivanja reči, inverzna frekvencija, veličina dokumenta.**

Jedan od čestih **poboljšanja** je model koji bolje definiše uticaj veličine dokumenta na relevantnost, jer sistem koji koristi BM25 favorizuje dokumenta manje veličine. Korišćenje sinonima i odsecanja sufiska/prefiska predstavlja neku vrstu poboljšanja ali najznačajnije poboljšanje donosi korišćenje **metapodataka**.To su ključne reči koje nisu vidljive u dokumentu.

Pojam **ekstrakcija informacija** odnosi se na **proces usvajanja znanja iz nestruktuiranog teksta na prirodnom jeziku** uočavanjem određenih klasa, objekata i veza između objekata. Jedna vrsta ekstrakcije informacija odnosi se na **ekstrakciju atributa nekog objekta** gde se pretpostavlja da se ceo tekst odnosi na jedan objekat pa se vade atributi tog objekta. Obično se postavlja neki **template-a korišćenjem regex-a**.

Jedan od čestih pristupa u ekstrakciji informacija je **Skriveni Model Markova** koji se zasniva na verovatnoćama skrivenih stanja.

25. Formalni jezici. Stohastička kontekstno-slobodna gramatika.

Formalni jezici (kao što su Java ili Python) **imaju precizno definisan model** i za svaki izraz možemo reći da li pripada tom jeziku ili ne. **Formalni jezici se specificiraju skupom pravila koji se nazivaju gramatika**. **Formalni jezici takođe propisuju pravila kojima se izrazima dodeljuje smisao, odnosno semantika**.

Neki formalni jezik **možemo predstaviti u formi stabla** gde čvorove koji imaju decu obeležavamo kao **neterminalne simbole** a listove stabla kao **terminalne** tj. završne simbole.

Gramatiku formalnog jezika čine pravila oblika $\propto \rightarrow \beta$ gde $\propto$ i β mogu da sadrže terminalne i neterminalne simbole. Na osnovu restrikcija na sadržaj $\propto$ i β postoje:

- Konteksno-slobodne gramatike (neterminali -> bilo šta)
- Konteksno-osetljive gramatike (Desna strana sadrži makar onoliko terminala koliko ih ima na levoj)
- Regularne gramatike (neterminal -> terminal[neterminal])

Leksikon je lista reči dostupnih u jednom jeziku koje grupišemo po kategorijama (imenice, zamenice, pridevi, glagoli, brojevi).

Popularni model jezika u problemima obrade prirodnog jezika je **stohastička kontekstno-slobodna gramatika.** Podrazumeva specifikaciju verovatnoća u pravilima gramatike.

GD -> Glagol[0.70] | GD ID [0.30] Značenje verovatnoća u pravilu je da se glagolski deo sa verovatnodom 0.7 sastoji od glagola, a sa verovatnoćom 0.30 od složenije strukture koja sadrži glagolski i imenski deo.

26. Sintaksna analiza. Gramatika i logika.

Sintaksna analiza teksta odnosi se na proces parsiranja kojim se tekst analizira i prepoznaje strukturu definisanu gramatikom.

Postoje dve vrste parsiranja to su **top-down i bottom-up.**

Kod top-down parsiranja **počinjemo od početnog neterminalnog simbola** (S) i u svakom koraku vršimo uparivanje elemenata sa levom stranom pravila (X). Ako pronađemo ovakvo pravilo zamenjujemo X sa Y.

Parsiranje buttom-up počinje od konkretne rečenice i u svakom koraku uparuje elemente sa desnom stranom pravila (Y) i ukoliko uparivanje uspe zamenjuje se Y sa X.

Parsiranja top-down i bottom-up su često **neefikasna pri obradi prirodnog jezika**. Da bi se izbegla ovakva moguća neefikasnost koristi se dinamičko programiranje sa algoritmom koji se zove **chart parser. Chart** je struktura podataka koja predstavlja analizirani sačuvani podstring.

Gramatike se mogu proširiti kako bi se neke konstrukcije vezale za kontekst. Takva proširena gramatika se može opisati logikom prvog reda koristeći DCG (definite clause grammar) gramatiku konačnih klauzula. Takve klauzule su disjunkcije literala u kojima je tačno jedan pozitivan.

Predstavljanje **pravila gramatike u logici prvog reda nam omogućava da parsiranje posmatramo kao logičko rezonovanje** i da obradu jezika vršimo pomoću pravila zaključivanja u logici. Pored parsiranja, logičko rezonovanje se može koristiti i za **generisanje jezika**.

Kraj !

Napomena: Za ocene 6,7 predlažem da se preleće ova skripta uz propratne prezentacije(Tema1, 2, ...) sa mudla. Za ocene više od 8 predlažem da se ipak produbi služeći se skriptom(Skripta2014) sa mudla.

Srećno polaganje!